嵌套的计算系统进程级自适应抗衰重启策略研究及模型分析*)

王 湛 赵颜利 刘凤玉 张 宏
(南京理工大学计算机系 南京 210094)

摘 要 执行细粒度的进程级软件抗衰可以进一步降低抗衰成本,提高软件可靠性。本文针对软件系统中进程间交互频繁多变且交互关系难以判定的特点,分析进程间控制、调用及数据访问的关系,重新定义了进程重启相关度,提出了自适应进程相关拓扑图的算法理论,从而制定了嵌套的进程级软件抗衰重启策略,并在此基础上构建了策略实施模型,从而为全面实现智能化的软件系统细粒度软件抗衰提供了支持。

关键词 软件抗衰,抗衰粒度,重启相关度,自适应

Research and Modeling Analysis on Adaptive Process-level Nested Software Rejuvenation Policy of Computing System

WANG Zhan ZHAO Yan-Li LIU Feng-Yu ZHANG Hong
(Department of Computer, Nanjing University of Science and Technology, Nanjing 210094)

Abstract Software rejuvenation with fine rejuvenation granularity can improve software availability and reliability and save the cost of software rejuvenation. Based on the characteristics of process which corresponds in high frequency and has complex corresponding relation. Based on coupling relation between the processes of the software systems, this paper defines the degree of restart dependence between processes in a new way, brings forward the arithmetic theory of self-adaptable mutuality topology charts of processes , sets down the nested software rejuvenation policy on process-level, and then models the software rejuvenation policy. So it can support to execute intelligentized software rejuvenation with fine rejuvenation granularity.

Keywords Software rejuvenation ,Rejuvenation granularity ,Degree of restart dependence, Adaptive

1 引言

软件抗衰(Software Rejuvenation)[1]的提出缘于对软件老化现象的认识和对系统可靠性的需求。软件老化是指在软件的长期不间断运行过程中,由于系统内存占用和泄漏、未释放的文件锁、数据更新不及时、存储空间碎片以及舍入误差的累积等而导致的软件性能的衰退[1~3]。软件老化最终导致软件的失效。而SR是当软件性能衰退到一定程度时,终止程序的继续运行,重启系统以清理其内部状态(如进行垃圾收集、刷新操作系统内核表、重新初始化内部数据结构等),从而释放操作系统资源,使软件性能得到恢复[2,3]。为了进一步降低抗衰成本,希望将细粒度的抗衰策略执行到进程级[3~6]。

计算系统软件抗衰重启技术研究中提出的重启树(restart tree)[9]和George Candea提出的微重启(Microreboot)[7,8]都是从模块执行恢复,没有从更细粒度的进程级执行恢复;同时策略必须基于软件体系结构十分清楚的前提下才能实行,智能性和适用性都相对较弱。要执行系统进程级的SR,可以借鉴文[8,9]的思想,首先分析系统进程交互特点,定义合理的进程重启相关度,进而制定合理的智能化判定系统进程间重启相关度的算法,从而最终制定出重启策略和重启模型。

本文根据软件系统进程运行交互的特点,分析了进程间控制调用及数据访问的关系,重新制定了进程重启相关度的判定依据,并在此基础上给出了自适应进程相关拓扑图的算法理论,大大降低了在复杂系统中判定进程间重启相关度的复杂度,实现了重启技术的智能化。由此指定了系统嵌套的进程级抗衰重启策略,并构建了重启实施模型,从而为实现智能化系统细粒度抗衰提供支持。

2 进程重启相关度

进程就是一个正在执行的程序,包括程序计数器、寄存器和变量的当前值。不同的软件系统皆由协同工作的进程构成。尽管每个进程是一个独立的实体,但进程间经常发生交互作用。

定义1 进程重启相关度是系统进程间互连的紧密程度,它是数据传输率、响应时间、并行处理能力等性能指标的综合反应,它主要取决于所选用的互连拓扑结构和通信链路的类,记为 $D[x \cdot y]$。分析系统进程间的耦合程度,将重启相关度取值为$\{-1,1,2,3,4\}$,特别地定义 $D[x \cdot x]$为进程 x 自身重启相关度,且 $D[x \cdot x]=4$。

假定系统的两个不同的进程 P_i 和 P_j,I_i、I_j 分别是进程 P_i 和 P_j 的输入变量集合,O_i、O_j 分别是进程 P_i 和 P_j 的输出变量集合。

$$I_j \cap O_i = \varnothing \tag{1}$$

*)国家自然科学基金项目名称(60273035)、国防科工委基础应用项目(K1704060511)。王 湛 博士研究生,主要研究方向:软件性能保持;赵颜利 博士研究生,主要研究方向:计算机视觉、图形与图像;刘凤玉 教授,博导,主要研究领域:人工智能与信息安全;张 宏 教授,博导,主要研究领域:人工智能与信息安全。

$$I_i \cap O_j = \varnothing \quad (2)$$

$$O_i \cap O_j = \varnothing \quad (3)$$

此时若三个条件都满足，则称进程 P_i 和 P_j 没有相关性，此时重启相关度为 0；若条件(1)不满足，则称进程 P_i 和 P_j 流相关，记为 $P_i \delta P_j$，此时 $Da[P_i \cdot P_j]=1$；若条件(2)不满足，则进程 P_i 和 P_j 反相关，记为 $P_i \bar{\delta}^0 P_j$，此时 $D[P_i \cdot P_j]=-1$；若条件(3)不满足，则称进程 P_i 和 P_j 输出相关，记为 $P_i \delta^o P_j$，此时 $D[P_i \cdot P_j]=2$；若 O_i 中的元素出现在 P_j 中的控制语句中，则称 P_i 依赖于 P_j 的控制相关，记为 $P_i \delta^c P_j$，此时 $D[P_i \cdot P_j]=3$；若进程 P_i 和 P_j 使用了相同的稀缺资源，则称 P_i 与 P_j 资源相关，记为 $P_i \delta^r P_j$，此时 $D[P_i \cdot P_j]=0$。本文中将采取直接交互方式而发生控制相关、资源相关、流相关、反相关、输出相关五种耦合关系的进程称为直接相关进程。

性质 1 输出相关具有交换性和传递性。若 $P_j \delta^0 P_i$，$P_j \delta^0 P_k$，则 $P_i \delta^0 P_k$；若 $P_i \delta^0 P_j$，则 $P_j \delta^0 P_i$。

定理 1 若进程 P_i 和 P_j 输出相关，则其中一个进程重启，另一进程同时重启。

假定 $P_i \delta^0 P_j$；则进程 P_i 和 P_j 总是同时重启，因为 O_i、O_j 互相使用了对方数据，如果不同时重启，会导致数据不一致或数据冲突。

性质 2 流相关、控制相关具有传递性。若 $P_i \delta P_j$，$P_j \delta P_k$，则 $P_i \delta P_k$；若 $P_i \delta^c P_j$，$P_j \delta^c P_k$，则 $P_i \delta^c P_k$。

定理 2 若进程 P_i 和 P_j 流相关或控制相关，则进程 P_i 重启，P_j 同时重启；P_j 重启时，P_i 不一定重启。

假定 $P_i \delta P_j$，则当进程 P_i 重启时，P_j 必须同时重启，否则 I_j 不能与 O_i 的状态保持一致；而当进程 P_j 重启时，P_i 则不一定重启，因为 P_j 并不向 P_i 传递控制和调用参数，不会影响其状态。同样可证，当 $P_i \delta^c P_j$ 时，重启时同样必须满足此定理。

性质 3 资源相关具有交换性。若 $P_i \delta^r P_j$，则 $P_j \delta^r P_i$。

当两进程非直接相连时，判定它们的重启相关度需要考虑其间经过的所有进程间的重启相关度。

约定 1 当两个进程之间存在多种重启相关度时，取其最大值。当两个进程之间存在多种相关性时，取程度最高的相关。这样可以保证重启的完善性和有效性。

定义 2 若进程 P_i 和 P_j 非直接连接，若 P_i 经若干进程可达 P_j，则认为从 P_i 到 P_j 间有一条可达路径 $R[P_i \sim P_j]$，经历的进程以“-”连接。

约定 2 若从进程 P_i 到 P_j 存在可达路径 $R[P_i \sim P_j]=P_i—P_1—\cdots P_n—P_j$，则进程 P_i 与 P_j 重启相关度为 $D[P_i \cdot P_j]=\mathrm{Min}\{D[P_i \cdot P_n], D[P_n \cdot P_j]\}$。

由以上分析，要确定进程间的重启相关度，需以下步骤：

(1)确认初始进程和终结进程。因为可达路径是单向的。

(2)查找从初始进程到终结进程的所有可达路径。

(3)按每一条可达路径求初始进程和终结进程间的重启相关度。

(4)确定初始进程和终结进程的最终重启相关度。

3 自适应进程相关拓扑图算法

系统实际运行时，进程间的交互往往是瞬息万变的，其可预测性很低，通过事先分析软件体系结构来确定进程间的重启相关度的方法其可行性和实用性都是很低的。而文[8,9]的思想都是基于进程间的交互是实施策略前可预测的，而实际情况往往是相背的；同时系统中进程数目通常较大，故制定智能化的进程重启相关度的判定方法迫在眉睫。

3.1 进程相关拓扑图

确定任意进程的重启相关度是系统重启策略实施的前提，针对系统中进程间交互频繁多变的特点，采用进程相关拓扑图来表示进程间的重启相关度是最简单清晰的方法。制定相关图统一表示如下：

假定系统的两个不同的进程 P_i 和 P_j，若 $P_i \delta P_j$，即 $D[P_i \cdot P_j]=1$，则由图 1 表示；若 $P_i \bar{\delta} P_j$，即 $D[P_i \cdot P_j]=-1$，则由图 2 表示；若 $P_i \delta^0 P_j$，即 $D[P_i \cdot P_j]=2$，则由图 3 表示；若 $P_i \delta^c P_j$，即 $D[P_i \cdot P_j]=3$，则由图 4 表示；若 $P_i \delta^r P_j$，即 $D[P_i \cdot P_j]=0$，则由图 5 表示。由此可得到离散图 $G=(V,E)$，其中 V 是进程顶点集合，$E=\{e_{ij} \mid e_{ij}=(V_i,V_j)V_i,V_j \in V\}$ 代表进程间有向边集，每条边由一个代表权值的非负权重 $W=\{W_{ij} \mid W_{ij}=W(V_i,V_j),W_{ij}>0,V_i,V_j \in V\}$。

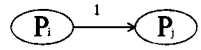

图1

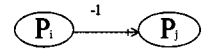

图2

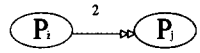

图3

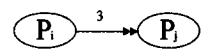

图4

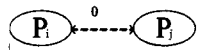

图5

3.2 自适应拓扑图算法理论

在系统进程拓扑图中，邻居的定义是进程节点之间可连接的(link-level)，即两进程顶点间边的权值大于零。此时各进程向自己的邻居进程报告与其相邻的链路状态，由这些针对某个进程的链路组成该进程的进程相关拓扑图。一个进程的相邻链路和其邻居报告的拓扑图形成了系统部分进程相关拓扑图，进程利用这个拓扑图来最终生成自身的进程相关拓扑图。此时当进程发现新的交互链接发生或者某个交互停止以及交互的变化导致通信延时过大时，则向邻居发送自身的进程相关拓扑图的更新信息。

执行此算法时，进程根据自身到达目的进程的路径连接状态发出更新信息，进而确立进程相关拓扑图。每个进程根据其通信交互状态和由邻居报告的源相关拓扑图共同形成了对该进程为已知的部分相关拓扑结构，此时只需更新源相关拓扑图，而不需通知正交互的和停止交互的邻居进程。因为每个目的进程只有一个先辈(predecessor)，进程只需更新其部分连接和惟一的子图表项。由此大大降低了算法的复杂度，同时充分展示了算法的自适应性。

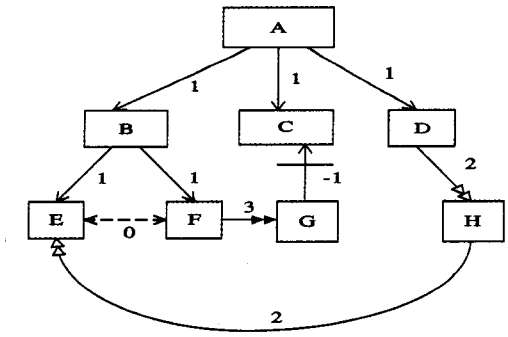

图 6 系统直接相关进程交互图

自适应进程相关拓扑图算法可以充分解决由于软件系统

进程间交互频繁多变、瞬时性很强而造成的进程间重启相关度判定算法时间和空间复杂度相对较高的问题；它充分利用了系统进程交互的特点，将其充分转化为算法制定的理论依据，使其转化为算法的特点和优势，使算法性能大大优于按需算法。

现假定系统中进程直接相关图如图 6 所示，则由上述算法可得系统进程间的重启相关度如表 1 所示。

表 1　图 6 中任意进程间重启相关度

重启相关度 / 初始进程 \ 终结进程	A	B	C	D	E	F	G	H
A	4	1	1	1	1	1	1	1
B	−1	4	0	1	1	1	1	1
C	−1	0	4	0	0	1	1	0
D	−1	−1	0	4	2	0	0	2
E	−1	−1	0	2	4	0	0	2
F	−1	−1	−1	0	0	4	3	0
G	−1	−1	−1	0	0	3	4	0
H	−1	−1	0	2	2	0	0	4

4　建立抗衰策略的实施过程模型

为了制定简约明确的重启抗衰策略实施模型，使其易于理解和分析，必须分析 SPN 以及 DFA 两种形式化描述方法的特点，综合了两种方法优点：用 SPN 描述每次抗衰的具体实施过程，用 DFA 控制各次抗衰的返回位置，明确描述各抗衰子过程之间的关联与转移，这样既避免了模型状态空间爆炸问题又避免了采用 DFA 模型不能描述策略的实施细节的缺点，使重启模型简单而明确，容易理解和分析。

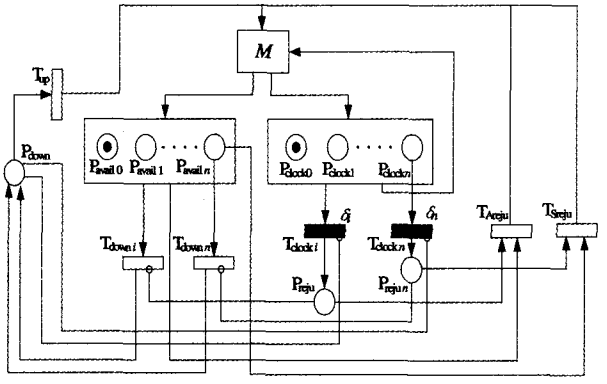

图 7　嵌套进程级抗衰重启策略的实施过程模型

图 7 中两大矩形框包含两组位置：{$P_{avail\,0}$，$P_{avail\,1}$，…，$P_{avail\,n}$}和{$P_{clock\,0}$，$P_{clock\,1}$，…，$P_{clock\,n}$}，分别表示各抗衰子过程的初始位置和记录时钟，每次重启各有其中的一个位置参与。前 n 次进程级重启共用同一变迁 $T_{down\,i}$、$T_{clock\,i}$。但是，对应不同的初始位置，$T_{down\,i}$服从不同的随机分布 q_n，$T_{clock\,i}$的实施时间 δ_i($0\leqslant i<n$)也不同，进入抗衰状态后实施相同变迁 T_{Areju}，进程级重启。系统级重启时实施另一变迁 T_{Sreju}，使用不同的位置 $P_{reju\,n}$和变迁 $T_{down\,n}$、$T_{clock\,n}$。任一时刻若系统进入失效状态，实施相同的变迁 T_{up}，服从相同的随机分布。

标有 M 的矩形框表示的有限状态自动机如图 8 所示，它用以控制每次重启或重新引导后两大矩形框中标记进入的位置。标记在哪个位置，哪个位置作为抗衰子过程的初始位置。具体方法如下：

用 M 的有限状态集 Q 中的状态 q_i 控制位置组{$P_{avail\,0}$，$P_{avail\,1}$，…，$P_{avail\,n}$}和{$P_{clock\,0}$，$P_{clock\,1}$，…，$P_{clock\,n}$}中的标记，定义状态值如下：

$$\begin{cases} q_0 = 1000\cdots00 \\ q_1 = 0100\cdots00 \\ \cdots\cdots \\ q_n = 0000\cdots\cdots01 \end{cases} \tag{1}$$

对应初始状态 q_0，标记位于 $P_{avail\,0}$和 $P_{clock\,0}$；进入状态 q_i ＝00…010…00(第 i 位为 1)，标记位于 $P_{avail\,i}$和$P_{clock\,i}$，此时系统已执行了 i 次进程级重启；到达状态 q_n，标记位于 $P_{avail\,n}$和 $P_{clock\,n}$，准备执行系统级重启。

从 P_{clock}位置组到自动机 M 的连接线，表示 M 的一个输入，由此获知系统的当前状态 q_i；从变迁 T_{Areju}、T_{Sreju}、T_{up}到自动机 M，三线合一作为 M 的另一个输入，变迁 T_{Areju}、T_{Sreju}、T_{up}的实施分别对应∑中的 a、s、r，由此获知系统执行的操作；M 的转移函数 θ 读取这两个输入，按公式(2) 给定的状态转移规则，得到系统的下一状态，作为 M 的输出，并按公式(1) 控制下一恢复子过程中标记的初始位置。

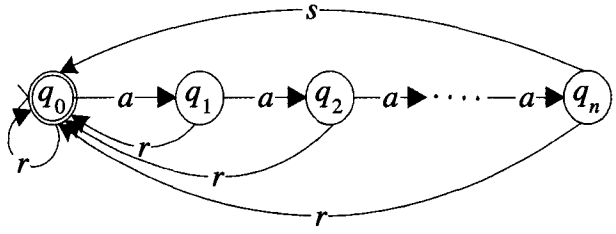

图 8　嵌套进程级抗衰重启策略的系统 DFA 模型

图 8 中描述的状态转移规则可以简化为下式：

$$\begin{cases} \theta(q_i, a) = q_{i+1} & 0\leqslant i<n \\ \theta(q_n, s) = q_0 \\ \theta(q_i, r) = q_0 & 0\leqslant i\leqslant n \end{cases} \tag{2}$$

式(2)表示系统自初始状态 q_0 经过 n 次进程级重启后执行系统级重启，重新进入初始状态；如果系统在运行过程中意外崩溃，则无论系统当前处于哪一状态，均要重新引导系统(r)，使其回到初始状态。

结论　要制定合理的智能化系统进程级的细粒度软件抗衰策略，进一步增强软件抗衰的适用性，更大程度地节省抗衰成本，以此提高软件的可靠性，必须具备三个前提条件：一制定进程间的重启相关度判定方法；二智能化判定系统中任意进程间的重启相关度；三是构建易于理解的抗衰重启实施过程模型，为全面实行此策略提供支持。

为实现第一个条件，本文分析了进程间控制、调用及数据访问关系，重新定义了进程重启相关度的算法依据，为实现智能化判定任意进程重启相关度提供理论基础。

软件系统进程间交互频繁多变、瞬时性很强，以往对进程间重启相关度的判定算法的时间和空间复杂度相对较高。为了实现第二个条件，必须充分利用了系统进程交互的特点，将其充分转化为算法制定的理论依据，使其转化为算法的特点和优势，由此提出了自适应进程相关拓扑图算法，使其性能大大优于按需算法，提高策略执行的效率，实现抗衰智能化。

要满足第三个条件必须分析各描述方法的优缺点。本文分析了 SPN 以及 DFA 两种形式化描述方法的特点；综合了

(下转第 282 页)

实验结果表明：该测试引擎能够尽快发现错误，定位错误位置，分析错误类型，提高测试覆盖率和测试精度。

结论和未来研究方向 本文针对基于SOA的Web services软件的特点，并充分利用测试领域的相关知识和方法，在此基础上提出了基于XML技术的自动化测试引擎的核心算法，该算法的关键在于对复合Web services之间关系进行了模型描述。提出了顺序关系、分支关系、选择关系、并行关系等五种基本关系，通过配置测试用例，自动提取测试数据，采用SOAP协议对Web services进行访问并记录、分析测试结果，最终完成Web services集成测试。该方法对Web services软件集成有较大的帮助。

随着分布式计算的深入，Internet在应用广度的发展，使得智能设备、手机、PDA等设备使用Web服务的范围逐渐扩大，对Web服务软件采用自动化测试方法已是大势所趋。本文所提出的测试方法还有一些不足，如何针对移动设备及WAP协议完成服务软件自动化测试将是下一步研究的工作。

参考文献

1 Bai Xiaoying, Dong Wenli, Tsai W T, et al. WSDL-based Automatic Test Case Generation for Web Services Testing. In: Proc. of the 2005 IEEE International Workshop on Service-oriented System Engineering (SOSE' 05), 2005

2 Tsai W T, Paul R, Song W, et al. Coyote: An XML-based Framework for Web Services Testing. In: Proc. of the 7th IEEE International Symposium on High Assurance Systems Engineering (HASE'02), 2002

3 Tsai W T, Paul R, Wang Y, et al. Extending WSDL to Facilitate Web Services Testing. In: Proc. of IEEE HASE, 2002. 171～172

4 Tsai W T, Chen Y, Cao Z, et al. Testing Web Services Using Progressive Group Testing. In: Proc. of the Advanced Workshop on Content Computing, 2004. 314～322

5 Tsai W T, Chen Yinong, Paul R, et al. Adaptive Testing, Oracle Generation, and Test Case Ranking for WebServices. In: Proc. of the 29th Annual International Computer Software and Applications Conference (COMPSAC), Edinburgh, July 2005. 101～106

6 姜瑛，辛国茂，单锦辉，等. 一种Web服务的测试数据自动生成方法. 计算机学报，2005，28(4)

7 黄陇，于洪敏，陈致明，等. 基于UML的软件测试自动化研究. 计算机应用，2004，24(7)

8 郭勇，邓波，衣双辉. 面向服务的网格软件测试环境. 软件学报，2006，17(11)

9 赵俊峰，谢冰，张路，等. 一种支持领域特性的Web服务组装方法. 计算机学报，2005，28(4)

(上接第267页)

两种方法，用SPN描述每次抗衰的具体实施过程，用DFA控制各次抗衰的返回位置，明确描述各抗衰子过程之间的关联与转移。这样既避免了模型状态空间爆炸问题又避免了采用DFA模型不能描述策略的实施细节的缺点。由此，由此得出了简约明确的系统嵌套的进程级重启抗衰策略实施模型，并以其易于理解和分析的优点，为全面实施软件系统智能化细粒度软件抗衰策略提供支持。

参考文献

1 Garg S, Moorsel A V, Vaidyanathan K. A Methodology for Detection and Estimation of Software Aging. In: Proceedings of the 9th International Symposium on Software Reliability Engineering, Paderborn, Germany, 1998

2 Huang Y, Kintala C, Kolettis N. Software Rejuvenation: Analysis, Module and Applications. In: Proc. of FTCS-25, Pasadena, CA, 1995

3 Castelli V, Harper R E, Heidelberger P. Proactive Management of Software Aging. IBM JRD, 2001, 45(2): 311～332

4 Hong Y, Chen D, Li L. Closed Loop Design for Software Rejuvenation. In: Workshop on Self-Healing, Adaptive and self-MANaged Systems (SHAMAN), New York, 2002

5 Xiea W, Hongb Y, Trivedi K. Analysis of a two-level software rejuvenation policy. Reliability Engineering and System Safety, 2005, 87(1): 13～22

6 Patterson D, Brown A, Broadwell P. Recovery Oriented Computing (ROC): Motivation, Definition, Techniques, and Case Studies. [Technical Report]. UCB/CSD-02-1175. UC Berkeley Computer Science 2002

7 Candea G, Fox A. Recursive Restartability: Turning the Reboot Sledgehammer into a Scalpel. In: 8th Workshop on Hot Topics in Operating Systems, Schloss Elmau, Germany, 2001

8 Candea G, Kawamoto S, Fujiki Y. Microreboot - A Technique for Cheap Recovery. In: 6th Symposium on Operating Systems Design and Implementation, San Francisco, CA, 2004

9 Candea G, Cutler J, Fox A. Improving Availability with Recursive Microreboots: A Soft-State System Case Study. Performance Evaluation Journal, 2004, 56(1-3): 213～248

10 王湛，游静，赵颜利，等. 基于访问关系的进程重启相关性判定. 计算机科学[J]，2006，33(9)

11 游静，徐建，张琨. 计算系统软件抗衰重启技术研究[J]. 信息与控制(已录用，待发表)

12 You Jing, Xu Jian, Zhao Xue-long, et al. Modeling and Cost Analysis of Nested Software Rejuvenation Policy. LNCS 3612, 2005. 1280～1289

13 You Jing, Xu Jian, Zhao Xue-long, 等. Modeling and Availability Analysis of Nested Software Rejuvenation Policy. In: IEEE SMC, 2005

14 Murthy C S R, Manimaran G. Resource Management in Real-time System and Network. Cambridge, MA: MIT Press, 2001